TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics and Interdisciplinary Studies

Algoritmy využívané v 'point and click adventure' hrách

Algorithms used in 'point and click adventure' games

STUDIJNÍ PROGRAM B2646 – INFORMAČNÍ TECHNOLOGIE
STUDY PROGRAMME B2646 – INFORMATION TECHNOLOGY

STUDIJNÍ OBOR 1802T007 – INFORMAČNÍ TECHNOLOGIE
STUDY BRANCH 1802T007 – INFORMATION TECHNOLOGY

BAKALÁŘSKÁ PRÁCE | BACHELOR THESIS

Autor práce | Author — Michal Bohuslávek
Vedoucí práce | Thesis supervisor — Ing. Radek Srb

LIBEREC 2013

Tento list nahraďte

originálem zadání.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Datum:

Podpis:

Abstrakt

Tato bakalářská práce se věnuje tvorbě algoritmů používaných v „point and click adventure“ hrách. Obecná část definuje samotný žánr těchto her a zaměřuje se na problematiku tvorby algoritmů, které je nutné využít pro jejich vývoj.

Dále práce obsahuje návrh řešení, kde autor rozebírá způsoby řešení jednotlivých dílčích částí. Tyto části lze rozdělit na dvě hlavní – logickou část pro herní logiku a grafickou část pro samotné vykreslování.

V části realizace autor popisuje, jakým způsobem implementoval nastíněná řešení pomocí jazyku Java s ukázkou zdrojového kódu u některých klíčových částí. Zprávu ukončuje popis přikladu, vytvořeného pomocí naprogramovaných algoritmů.

Klíčová slova

Vývoj her, „point and click adventure“ hry, Java, OpenGL, LWJGL.

Abstract

This bachelor's thesis deals with creating algorithms used in point and click adventure games. General section defines the genre of these games itself and focuses on issues of creating algorithms, which are crucial for the development of these games.

Further, the thesis contains concept of solution where author analyses ways of solving individual parts. These parts can be divided into two main parts – a logic part for the game logic and an graphic part for the drawing.

In the section of implementation the author describes which ways were used to implement outlined solutions in Java programming language. Samples of the source code are given in some key sections. At the end of this thesis an example, which was created using programmed algorithms, is described.

Keywords

Game development, point and click adventure games, Java, OpenGL, LWJGL.

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Radku Srbovi za podporu a konzultace, doc. RNDr. Pavlu Satrapovi, Ph.D. za zpracování LaTeX šablony, ve které píši tuto zprávu a v neposlední řadě rovněž mému bratrovi Davidu Bohuslávkovi za tvorbu grafických textur při realizaci příkladu.

Obsah

Seznam obrázků

Seznam zkratek

LWJGL	Lightweight Java Game Library
OpenGL	Open Graphics Library
OpenAL	Open Audio Library
OS	Operační systém (*Operating System*)
TTF	TrueType Font

Úvod

`Point and click adventure` hry vznikly z textových herních adventur, což byly jedny z prvních her vůbec [Tich11]. Vznik textových adventur se datuje již k roku 1972. Tyto hry fungovaly způsobem, že hra nejprve popsala situaci, v níž se hráč nachází, a pak očekávala od hráče nějakou interakci – textový vstup. Uživatel musel sám vymýšlet, jaké asi mohl vývojář vyžadovat příkazy (např. `Get keys`).

S nástupem grafických aplikací tento žánr povýšil na grafickou úroveň. Popis prostředí textem byl nahrazen popisem pomocí grafických textur a hráč již nemusel vymýšlet a zkoušet různé příkazy, ale vystačil si s myší – odtud **point and click**. Tento žánr byl velmi oblíbený a rozšířený z důvodu propracovaného příběhu – byly to v podstatě jediné hry, které nějaký příběh měly – a také kvůli lépe zpracované grafice. Ta byla vypracovanější, protože v těchto hrách bylo zpravidla minimum pohyblivých objektů a nemusela se vykreslovat nějaká složitá 3D scéna, záleželo tedy spíše na šikovnosti herního grafika.

Motivací pro mě bylo oživení tohoto pomalu mizícího žánru (v dnešní době spíše vládnou 3D akční hry, zatímco point and click adventure hry pomalu ztrácí vývojáře i fanoušky) s cílem nabídnout případným zájemcům rozšiřitelné knihovny komplexních řešení rozličných problémů při vývoji podobných her. Bakalářská práce se zajímá jak o problémy sestavovaní herní logiky, tak o vykreslení grafických komponent.

Tato bakalářská práce si klade za cíl vyvinout kolekci algoritmů, které umožní či alespoň usnadní vývoj *point and click adventure* her. V této práci budu klást důraz na:

- Silný nástroj pro popis herní logiky[1].
- Nezávislost herní logiky na grafické reprezentaci.
- Nezávislost na operačním systému.

[1] Herní logika – příběh tvořený logickými hádankami, které jsou hlavním prvkem *point and click adventure* her.

Nástroj pro popis herní logiky umožní stručný, ale intuitivní zápis pro vytvoření herního příběhu. Při tvorbě využiji rysů jazyku `Java` pro zkonstruování silného objektového návrhu, jenž bude navržen takovým způsobem, aby byl dále rozšířitelný případnými pokračovateli této práce.

Nezávislost herní logiky na grafické stránce umožní využití oné části jako samostatné knihovny, připravené implementovat do jiného grafického rozhraní.

O nezávislost na OS se v první řadě postará jazyk `Java`, který je principiálně na této tezi založen, přičemž já se budu co nejvíce snažit tuto nezávislost podpořit.

1. Obecně o problematice

Problematika žánru *point and click adventure* her lze rozdělit do několika rovin. Podstatnou část tvoří herní logika, která definuje průběh a struktury hry, načež druhou částí, kterou lze rozdělit do několika podčástí, je grafická reprezentace hry.

1.1 Definice žánru

Úkolem hráče je vést herní postavu skrze prezentovaný příběh. K úspěšnému dokončení hry musí hráč procházet herními **lokacemi**, sbírat a kombinovat **předměty** a získávat informace od ostatních herních **postav**.

1.1.1 Definice pojmů

- `Lokace` – jedna herní obrazovka, místo, kde se nacházejí jisté postavy, předměty, východy…
- `Postava` – osoba, se kterou je možné komunikovat nebo jiným způsobem interagovat. Speciálním druhem postavy je postava hlavního hrdiny, kterého může hráč ovládat.
- `Hlavní postava` – osoba, kterou hráč ovládá a hraje její roli.
- `Předmět` – objekt, se kterým lze v lokaci nějakým způsobem reagovat (sebrání, použití s jiným předmětem…).
- `Východ` – slouží jako přechod mezi jednotlivými lokacemi.
- `Konverzace` – probíhá mezi postavami, je strukturovaná do výčtu dialogů.
- `Interakce` – jakákoli akce vyvolaná např. **sebráním předmětu, rozhovorem, kombinací předmětů…**

1.1.2 Popis herních postupů

Zkoumání a hledání objektů či východů hráč provádí naváděním myši po herní lokaci, dokud na nějaký objekt nenarazí. Přičemž hra dá takto najevo většinou změnou kurzoru myši či zobrazením nějakého textu, který hráče informuje o tom, co lze s objektem provádět. Pokud jsou objekty příliš malé, může se stát, že si jich uživatel nevšimne, což může způsobit komplikace při procházení hrou. Takovému jevu se pak říká **pixel hunting**[1].

Sbírání předmětů probíhá po kliknutí na příslušné předměty v herní lokaci. Pokud je zvolený předmět určen k sebrání, objeví se v **inventáři** herní postavy. Inventář mívá neomezenou kapacitu.

Používání předmětů nastane rovněž po kliknutí, pokud předmět není určen k sebrání. Většinou poté dojde ke změně stavu (*např. otevření zipu*).

Ke kombinování předmětů se používá inventář. Po kliknutí na **předmět** z inventáře se zvolený předmět označí jako aktivní a umožňuje kombinaci s ostatními předměty z inventáře nebo s předměty z herní lokace. Kombinovat lze v jeden okamžik pouze **2 předměty**, přičemž kombinací může vzniknout:

- `jiný předmět` – *např. kombinací pazourku a topůrka vznikne sekera*,
- `změna stavu` – *např. kombinací klíče na zámek dojde k odemčení zámku*,
- případně jiné interakce.

Konverzace s postavami probíhají po nějaké **interakci** (nejčastěji po kliknutí na postavu, se kterou chce hráč konverzovat). Hra poté nabízí případný výběr z několika **dialogů**, takže komunikace s postavou není statická – hráč si sám volí pořadí jednotlivých témat.

1.2 Problematika algoritmů

1.2.1 Herní logika

Nejprve popíši problematiku logické struktury hry, která tvoří značnou část vývoje her tohoto žánru. Největší úskalí spočívá v tom, že prakticky **neexistuje** žádný definovaný postup, jak herní logiku tvořit. Algoritmy pro vytvoření herní logiky

[1] Šli jsme na pixely. KLEKNER. *Hrej.cz* [online]. 2007 [cit. 2013-05-07]. Dostupné z: http://pc.hrej.cz/clanky/sli-jsme-do-lesa-na-pixely-xx-3672/

jsou použity jen u komerčních her nebo freewarových, ale nikoliv opensourcových her.

Veškerými zdroji při postupu vyvíjení herní logiky mi byly zkušenosti s herními tituly, které jsem kdysi hrál. Mezi ty patří určitě **Polda 4**, **Posel smrti**, **Horké léto** či **Runaway: A Road Adventure**. Mezi zjištěné skutečnosti patří:

Objekty[2] lze rozdělit do několika kategorií:

- `Postava` – viz kapitola 1.1.1.
- `Východ` – viz kapitola 1.1.1.
- `Předmět` – viz kapitola 1.1.1.
  - `„Sebratelný" předmět` – předmět, který lze sebrat (*přesunout do inventáře*).

Pro uchování a řízení závislostí se využívá nějaké logické struktury, který obsahuje určité **logické uzly**, které na sobě závisí. Závislost v tomto kontextu znamená, že nelze uskutečnit určitou akci, dokud není splněno něco jiného předtím. *Příklad: nelze rozsvítit světlo, dokud není namontovaná žárovka.*

`Interakce` **(viz kapitola 1.1.1)** je vyvolána kliknutím na jakýkoli **objekt** a jedná se o tyto úkony:

- **Sebraní zdrojového předmětu** – přesun předmětu do inventáře.
- **Umístění předmětu** z inventáře.
- **Použití předmětu** z inventáře.
- **Vznik nového předmětu** – přidání nového předmětu do inventáře (*nastane např. kombinací jiných předmětů*).
- **Přemístění** – změna lokace.
- **Uskutečnění konverzace** – zahájení rozhovoru s nějakou postavou.

1.2.2 Grafická reprezentace

Grafická reprezentace už neobsahuje tolik vyhraněných postupů, které by existovaly jen pro tento žánr. Vykreslení lze rozdělit do několika částí:

- Vykreslení všech vrstev pozadí lokace.

[2] Objekty – veškeré aktivní prvky ve hře.

- Vykreslení předmětů lokace.
- Vykreslení hlavní postavy.

Pokud bychom nebrali v úvahu hlavní postavu, která je schopna se v lokaci nějak pohybovat (*většinou po kliknutí myší na určité místo si postava najde cestu k cílovému bodu a dojde tam*), dalo by se na tuto hru nahlížet jako na čistě 2D implementaci grafických objektů. Přes pozadí lokace by se překreslovaly jednotlivé předměty, tedy jen takové, které by bylo možné posléze sebrat nebo které by měnily stav, aby mělo smysl je vykreslovat. Statické předměty, které nijak nemění své grafické vlastnosti (*např. stůl, na který lze jen pokládat nějaké věci*) není třeba zvlášť vykreslovat.

Přítomnost postavy ovšem přináší nový pohled na vykreslování. Skutečnost, že se postava může v prostoru pohybovat, vyžaduje minimálně nějakou emulaci **3D prostoru**. Musí se měnit velikost postavy podle perspektivy (*pokud postava stojí někde vzadu, musí být menší než na předních pozicích v lokaci*). Jednotlivé předměty lokace je nutné vykreslovat před respektive za postavou podle toho, zda-li postava je za respektive před daným předmětem. Musí se rovněž řešit algoritmus pro hledání cesty postavy z počátečního bodu do cílového bodu (*postava musí obcházet předměty, které jí stojí v cestě*).

Řešení pomocí 3D scény

Nabízí programově nejjednodušší řešení s vykreslováním předmětů, jelikož o viditelnost a perspektivu se postará grafická knihovna. Toto řešení je v současnosti nejrozšířenější, protože dnešní počítače nemají větší problémy s vykreslováním 3D prostoru a přitom tento návrhový vzor umožňuje rozdělit jednotlivé části vykreslení scény mezi různé grafické týmy:

- Vytváření 2D textur
- Vytváření 3D modelů
- Skládání 3D modelů do scény

Toto řešení by zahrnovalo vytvoření 3D scén pro jednotlivé **lokace**, grafická knihovna [TTSRH02] poté nabízí prostředky pro identifikování a výběr objektu podle zadaných **x, y** souřadnic a úhlu natočení kamery.

Nespornou výhodou tohoto řešení je možnost natočení scény z více pohledů, nebo dokonce možnost otáčení scény přímo v samotné hře. Co se týče hlavní postavy, řešení umožňuje několik alternativ:

- lze vytvořit 3D model postavy a její pohyb řešit standardní cestou – po kliknutí nasměrovat postavu na žádané místo,
- celou hru koncipovat z pohledu první osoby a umožnit tak hráči pohyb ve scéně jako v akčních FPS[3] hrách.

Viditelnost objektů by řešila samotná grafická knihovna [TTSRH02], která umí dle pozic jednotlivých objektů v prostoru vykreslit scénu tak, aby bylo vidět jen to, co má být vidět.

Hledání cesty by se řešilo – pokud by hra nebyla koncipována z pohledu první osoby (*hráč by se pohyboval v prosotru zcela volně – sám by hledal cestu*) – pomocí `path-finding` algoritmů[4], kde počáteční bod by byla pozice postavy a cílový bod pak místo, které označil uživatel.

Simulování 3D scény pomocí 2D

Toto řešení se využívalo především v dřívějších dobách, neboť pro něj nebyl potřeba takový výpočetní výkon. Přináší však řadu komplikací, a sice že si herní vývojář musí vyřešit jednotlivé dílčí problémy (*viditelnost objektů, pseudo-perspektivu…*) sám.

Jak jsem již zmínil výše, všechny tyto problémy je nutné řešit jen kvůli implementaci hlavní postavy. Nejprve se však zaměřím na vykreslování ostatních objektů mimo hlavní postavu.

Pro detekci kolize – zjištění, zda-li je na daný objekt najeto kurzorem myši, lze použít 2 metody:

1. `Obdélníková kolize` – objekt by byl ohraničen **4 body** – *tedy obdélníkem* –, pomocí kterých lze jednoduše zjistit, je-li kurzor myši uvnitř obdélníku či nikoliv. Výhodou je jednodušší implementace a menší časová náročnost, nevýhodou pak horší nasazení na rozmanité tvary objektů.
2. `Polygonová kolize` – objekt by byl ohraničen **n body** – *tedy polygonem*[5]. Implementace algoritmu, který zjišťuje, zda-li je bod uvnitř či vně polygonu, je sice náročnější, avšak tato metoda lépe umožňuje popsat tvar objektu.

[3] FPS – First-Person shooter, simulace pohledu herní pohledy z vlastního pohledu (pohled z první osoby).

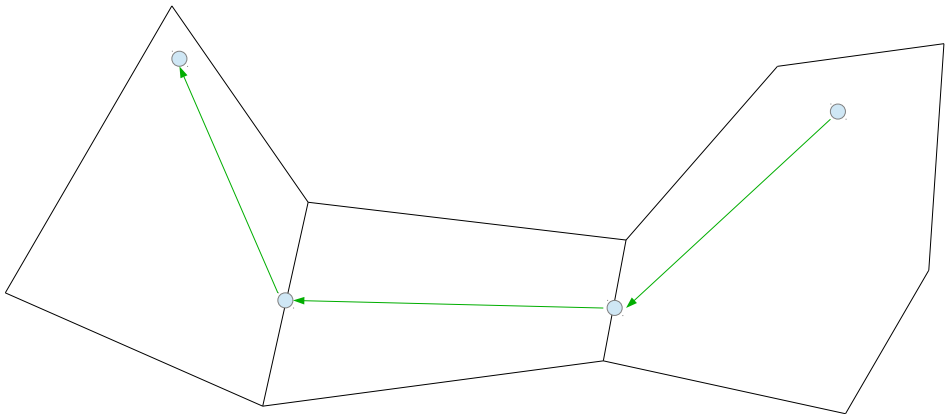
[4] Path-finding algoritmy – algoritmy určené pro hledání cesty z počátečního do cílového bodu dle určitých podmínek.

[5] Polygon – mnohoúhelník neboli n-úhelník.

Nehledě na zvoleném řešení by každý objekt měl k dispozici vlastnost **z-index**[6], kterou by se dalo ovlivnit, který objekt je navrchu v případě, že by se překrývalo více objektů.

S implementací postavy poté vyvstává více problémů:

Hledání cesty – tedy zamezení, aby postava na cestě z jednoho bodu do cílového bodu neprocházela skrze jiné objekty – lze implementovat tak, že oblast určená pro pohyb postavy bude tvořená konvexními polygony[7], které na sebe budou navazovat stranami. Pro pár dotýkajících se stran by pak byl zvolen bod, který by určoval průchod z jednoho polygonu na druhý. Při hledání cesty by se tedy hledala možná cesta z počátečního polygonu na cílový.



Obrázek 1.1: Ukázka řešení hledání cesty z jednoho bodu do druhého.

Viditelnost postavy lze řešit tak, že každý jednotlivý polygon bude mít také k dispozici vlastnost **z-index**, takže podle toho, na kterém polygonu postava stojí, lze určit, které objekty vykreslit za postavou a které před postavou. Je nutné mít však na paměti, že *z-index* by určoval hloubku pro celý polygon – tzn. musel by se zvolit tak, aby postava stojící na libovolném bodu polygonu měla před sebou respektive za sebou konstantní množinu objektů.

Perspektiva postavy je jedna z vlastností, která se nejhůře emuluje, protože prostor lokace může být více či méně rozmanitý se spoustou nakloněných rovin a různě vysokými místy pro pohyb postavy. Možnosti řešení jsou následující:

Primitivní řešení problému – každá lokace by měla jednu rovinu, což by sice dost omezovalo možnosti návrhu lokací, avšak výpočet velikosti postavy

[6] Z-index – vyjádření, jak hluboko je objekt v prostoru.

[7] Konvexní polygon – polygon, jehož vnitřní úhly jsou menší než *180°*.

by mohl být určen podle jednoduché rovnice,

$$h = \frac{a}{d} \tag{1.1}$$

kde h je zdánlivá výška objektu, a je skutečná výška objektu a d je vzdálenost objektu. Skutečná výška objektu by byla konstantou, kdežto vzdálenost objektu by se určovala podle **y souřadnice**.

Srovnání obou řešení:

Vlastnost	3D	2D
řešení viditelnosti	snadné	komplikované
perspektiva postavy	snadné	komplikované
kolize objektů	složitější	poměrně snadné
zpracování grafických podkladů	náročnější	snadné
variabilní pohled na scénu	ano	ne

Tabulka 1.1: Srovnání řešení pomocí 3D a 2D

2. Návrh řešení

2.1 Řešení logické části

Celý koncept logické části se zdál být velmi obsáhlý, přičemž bylo velice obtížné dát dohromady alespoň kostru návrhu. Vzhledem k tomuto problému jsem si nejprve celou část rozdělil na několik menších části, které jsem sice nemohl považovat za ryze samostatné, jelikož jsou mezi sebou více či méně propojené, nicméně vývoj těchto dílčích částí se již nezdál tak neřešitelný.

Po dokončení jednotlivých návrhů jsem teprve mohl zjistit, kde jsou úskalí vazeb mezi částmi, abych je mohl předělat do funkčního modelu. Tento proces však zahrnoval dlouhodobé experimentování, dokud jsme nedospěl ke konečnému návrhu.

Co se týče samotného rozdělování na dílčí části, vyhodnotil jsem ze zkoumání, že herní logika bude tvořena `předměty`, `postavami`, které mohou být schopné nějaké `konverzace`, dále `průchody` do jiných lokací a místem pro ukládání stavů, což jsem nazval `logickými uzly`.

Rozdělení dílčích částí problému bylo následující:

- logický uzel,
- předměty, postavy a východy,
- kombinování předmětů,
- řešení rozhovorů.

2.1.1 Logický uzel

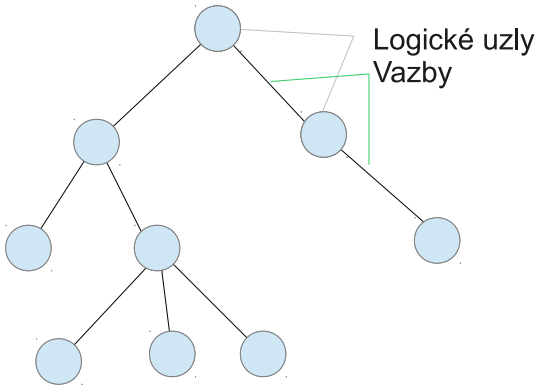
Začal jsem návrhem logického uzlu, jelikož se mi zdál implementačně nejméně závislý na fungování ostatních částí. Z pozorování vyplynulo, že logický uzel nemůže být splněn, dokud nejsou splněné všechny uzly na něm závislé. Jednotlivé úkony (*např. rozhovor, sebrání předmětu, kombinace předmětu…*) pak mohou být vázané na tyto

logické uzly, a to ve smyslu, že se pokusí nějaký uzel splnit (**zápis**) nebo podle stavu určitého uzlu (**čtení**) rozhodnou o příslušném úkonu.

Má původní představa byla taková, že logický uzel bude svázán s **libovolným počtem podřízených uzlů** (*potomků*), na kterých bude závislý, a bude rovněž vázán na **jeden nadřazený uzel** (*předek*), na němž bude závislost vyžadovat. Tímto by se vytvořila stromová datová struktura[1].

Každý jednotlivý uzel by obsahoval informace, zda-li je **proveden** a zda-li je **připraven** (všechny jeho podřízené uzly byly provedeny). Dále by obsahoval nástroje pro provedení – uzel by bylo možné provést pouze v případě, že je připraven.

Uzel by byl implicitně ve stavu `připraven` pouze v případě, že by se jednalo o list[2] struktury. Po provedení uzlu by uzel o tomto faktu informoval svého předka, který by si ověřil stav svých potomku a v případě, že by byly všechny provedeny, nastavil by se do stavu `připraven`.



Obrázek 2.1: Ukázka logického stromu – původní návrh

Tento návrh jsem posléze označil za nevhodný ze 2 důvodů:

1. logický uzel může vytvářet závislost pro více předků (*tedy 1 předek nestačí*),
2. je zbytečné, ba dokonce implementačně nevhodné, aby byly rozlišovány stavy `připraven` a `proveden`.

Po opravení těchto nedostatků došlo k finální podobě logického uzlu:

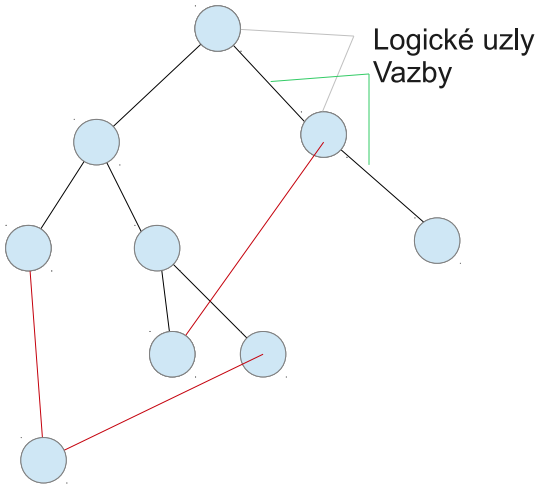
Libovolný počet předků změnil původní stromovou strukturu na trochu komplikovanější, což však nevedlo k žádným komplikacím.

Libovolný počet potomků zůstal zachován.

[1] Strom – datová struktura tvořena neorientovaným grafem, který je souvislý a neobsahuje žádnou kružnici.

[2] List – uzel stromu, který již neobsahuje žádné potomky (*podřízené uzly*).

Vlastnost splněn nahradila původní vlastnosti `proveden` a `připraven`. Nadřazené uzly se tedy splní (*provedou*) automaticky v okamžik, kdy se splní všechny podřízené uzly. Explicitně má tedy smysl plnit pouze koncové uzly, které nemají žádné potomky.



Obrázek 2.2: Ukázka finální logické struktury

2.1.2 Objekty – předměty, postavy a východy

Aktivními prvky lokací jsou právě **předměty**, **postavy** a **východy**. Při bližším zkoumání jsem však dospěl k závěru, že rozdíly mezi těmito jednotlivými prvky nejsou nikterak markantní, v zásadě jde tedy jen o to dát hráči informaci, co by měl očekávat, jak bude daný prvek reagovat – *předmět bude sbírat nebo nějak používat, s postavou bude vést konverzaci, východem se dostane do jiné lokace*) – ačkoli to tak být vůbec nemusí.

Návrh tedy vedl k tomu všem těmto prvkům dát nějaké společné vlastnosti a definovat jim předka, od kterého tyto vlastnosti budou dědit. Tento předek budiž nazýván `Objekt`, přičemž jednotlivé lokace obsahují všechny tyto objekty na jednom místě nehledě na to, jaký je jeho skutečný typ.

2.1.3 Kombinování předmětů

Už od začátku jsem přemýšlel nad tím, jak implementovat kombinování předmětů, tedy např. kombinací `klíče` a `dveří` se `odemknou` dveře nebo kombinací `mušlí`

a `provázku` vznikne `náhrdelník`. Přemýšlel jsem, jakou strukturu navrhnout, aby tohoto byla schopna.

Pozorováním a přemýšlením nad vzorovými situacemi jsem si dále všímal dalších skutečností:

- Kombinovat lze mezi 2 předměty z inventáře nebo mezi předmětem z **inventáře** a předmětem z **lokace**. Každopádně se vždy jedná o kombinaci **2 předmětů**.
- Při kombinovaní může nastat škála různých situací. Kombinace může:

  způsobit ztrátu původních předmětů výměnou za jeden nový (*z provázku a mušle se stane náhrdelník*),

  pouze splnit logický uzel (*odemčení dveří klíčem*),

  pouze uskutečnit rozhovor a vysvětlit hráči, že tuto věc nemůže provést (*nelze vystřelit z nenabité zbraně – navádění hráče k správnému řešení*),

  zničit či zneaktivnit předmět z lokace, …
- Podobné úkony jako při kombinování předmětů mohou vzniknout také při pouhém aktivování předmětu z lokace – *kliknutí na předmět* – nebo při rozhovoru s postavou. Z toho vyplývá, že vykonání akce není závislé na typu zdroje – kliknutím na postavu (*zahájením konverzace*) lze vyvolat přesun do jiné lokace, stejně jako kliknutím na východ (*přesun do jiné lokace*) lze vyvolat rozhovor –, tedy objektu, co onu akci vyvolalo. Řešení by tudíž mělo být nějak sjednocené.

2.1.4 Interakce

Na základě zjištěných informací mě návrh nasměroval k vytvoření společného prvku řešení všech úkonů – **interakce**. Ta se stará o popis situace (*zachycení úkonů*), ke kterým má při provádění interakce dojít. Během vývoje jsem postupně přidával další typy úkonů, které nebyly znát již od začátku.

Mazání objektů a přidávání předmětů do inventáře

V první fázi jsem přidal mazání objektů a naplňování inventáře. Vycházel jsem ze situace:

Vzorová situace:

Zkombinováním pazourku a klacku se vytvoří sekera.

Při zkoumání této situace jsem se rozhodl vytvořit v interakci pole, kam se budou dávat objekty, které se mají přidat do inventáře, a pole, kam přijdou předměty ke smazání. V tomto vzorovém případě tomu odpovídá „smazání pazourku a klacku“ a „přidání sekery do inventáře“.

Tímto způsobem tedy lze řešit spoustu dalších situací jako např.:

- Sebrání předmětu z lokace – nejprve vymazat, poté přidat do inventáře.
- Vymazání předmětu z inventáře po použití (*např. záchranné pistole*).
- Přidání předmětu do inventáře po použití jiného předmětu (*např. odtržení lístku z pořadníku*)

Přemístění

Potřeba přemístění vznikla ve chvíli, kdy do příkladu hry, na níž probíhal vývoj, přibyla další lokace. V interakci tedy přibyla možnost nastavit odkaz na lokaci, do níž se má přesunout.

Umístění předmětu

Dále bylo potřeba umísťovat předměty z inventáře do cílové lokace.

Vzorová situace:

Přesunutí plachty z inventáře na rozestavěný stan.

Opět tedy přibyla možnost nastavit cílovou lokace, do níž se má předmět vložit.

Aktivace a deaktivace objektu

Jsou situace, kdy herní logika potřebuje měnit stav, zda-li je objekt aktivní nebo ne.

Vzorová situace:

Vchod do stanu je aktivní jen ve chvíli, kdy je otevřený zip.

Bylo tedy nutné předat do interakce „`aktivátor`“ a „`deaktivátor`“, což jsou odkazy na `logický uzel`, který pak objektům, je-li uzel splněn, nastavuje stav. Vyšší prioritu má deaktivátor, takže pokud je splněn, objekt je neaktivní, i když je aktivátor také splněn.

Uskutečnění konverzace

Uskutečnění konverzace se také stalo jen dalším úkonem v interakci, jelikož konverzace se ne vždy vyvolává jen kliknutím na osobu. Dále o konverzaci viz kapitola 2.1.5.

Dokončení hry

Vznikla rovněž potřeba nějakým způsobem vyhodnotit, že hráč dokončil hru, což bylo opět přesunuto do interakce. Poté může dojít k přesunu do menu, závěrečné animaci, titulky…

2.1.5 Konverzace

Konverzaci lze brát za relativně nezávislou součást. Už na začátku jsem si definoval několik požadavků:

- Konverzace může probíhat **mezi několika postavami** současně, počítá se však s tím, že v jeden okamžik mluví pouze jedna z nich.
- Konverzace může být rozdělena do několika oddělených dialogů.
- Pořadí dialogů není pevně stanoveno.
- Dialog sestává z několika „útržků řeči“, které jsou vždy přiřazeny postavě, která je pronáší.

Nejmenší prvek v konverzaci je „`útržek řeči`“. Při sepisování konverzace se vždy tento útržek přiřazuje k nějaké osobě, přičemž pořadí těchto útržků je chronologické podle toho, jak byly vloženy. Tyto útržky se přidávají jednotlivým **dialogům**. Konverzace má jeden **výchozí dialog**, který se spustí pokaždé při zahájení konverzace, a dále obsahuje pole dialogů, do kterého lze přidávat další dílčí dialogy.

Útržek řeči

Útržek řeči je koncipován tak, že se skládá z pouhého textové řetězce, pokud by byl brán v úvahu také dabing, měl by útržek obsahovat také odkaz na `audio stopu`.

Provázanost s `logickým uzlem`

I konverzaci bylo nutné provázat, a to hned ze 2 důvodů:

1. Jsou situace, kdy s postavou není hned od začátku možné hovořit o jisté věci (*konverzace zatím neobsahuje dialog o té věci*). Příklad:

   Nelze hovořit s postavou o zamčených dveřích, dokud hráč nezjistí, že jsou ony dveře opravdu zamčené.

2. Jiné situace zase vyžadují, aby konverzace ponechávala určitý dialog – standardní vlastností jednotlivých dialogů konverzace je totiž to, že se po provedení z konverzace odstraní –, dokud není něco splněno. Příklad:

   S krupiérem je možné hovořit o přijímání sázek, dokud hráč nevyhraje a nesplní tím úkol.

Každý dialog může být tedy svázán s **logickým uzlem**, aby se do konverzace přidal v určitý čas nebo naopak odebral, pokud již není potřeba.

Provedení `interakce`

Nejenže součástí interakce může být zahájení konverzace, nýbrž i konverzace samotná někdy vyžaduje spuštění nějaké jiné interakce.

Vzorová situace:

> *Po promluvení s postavou o klíči postava předá klíč hlavní postavě – klíč se přesune do inventáře.*

Dialog tedy obsahuje možnost nastavit interakci, která se má při provádění dialogu spustit.

2.2 Řešení grafické části

Vzhledem k tomu, že je logická část striktně oddělena od grafické implementace, záleží na vývojáři, pro jaké grafické řešení se rozhodne. Nicméně v této práce jsem se rozhodl řešit i grafickou část.

Zvolil jsem řešení pomocí **2D** z několika důvodů:

- **Subjektivní preference** – považuji hry tohoto žánru za zdařilejší, jsou-li zpracovány pomocí `2D`. Přidáním dalšího rozměru ztrácí své kouzlo.

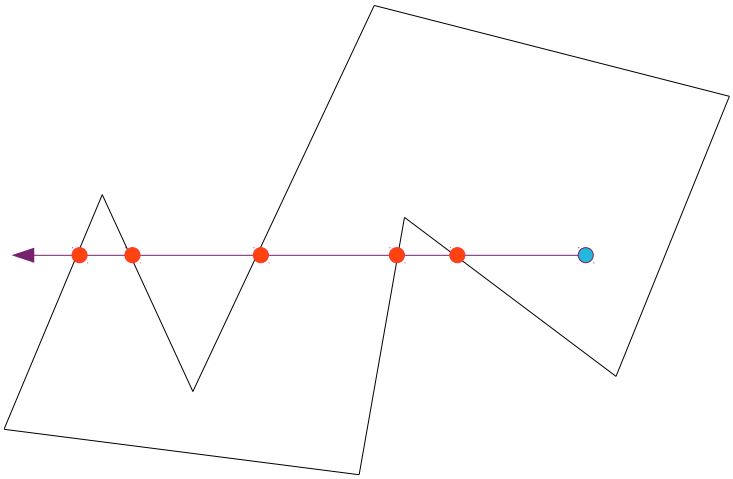
- **Minimální nutnost vytvoření editoru pro tvorbu lokací** – obrázky lokací a předmětů lze tvořit v kterémkoli grafickém editoru, není nutné řešit mapování textur a celkovou tvorbu `3D` prostoru.
- **Dostupný 2D grafik** – o tvorbu grafických textur se postaral můj bratr David Bohuslávek, čímž mi usnadnil vytvoření příkladové hry.

Detekce kolizí

Pro řešení kolizí – tedy zjištění, zda-li je na určitý objekt ukázáno kurzorem myši – jsem nejprve pro jednoduchost implementace zvažoval `obdélníkovou kolizi`, brzy jsem však z důvodu již zmíněných výhod přešel k `polygonové kolizi` (viz kapitola 1.2.2).

Každý objekt bude mít možnost přiřazení libovolného počtu bodů $(x,\ y)$ – **polygonu** –, které budou definovat jeho pozici v lokaci. Ze zadaných souřadnic bodů a ze souřadnic pozice kurzoru myši lze pak určit, zda-li je na objekt ukazováno. Ke zjištění této informace jsem řešil problém určení, zda-li bod leží vně či uvnitř polygonu.

K tomuto účelu jsem vybral algoritmus `průchod paprsku` [Fin07], který funguje na principu vyslání paprsku z bodu, u kterého chceme zjistit polohu (*zda vně či uvnitř*), a počítání počtu průchodů stěnami polygonu. Pokud je počet průchodů lichý respektive sudý, leží onen bod uvnitř respektive vně polygonu.



Obrázek 2.3: Princip určení, leží-li bod uvnitř či vně polygonu.

Perspektiva postavy a hledání cesty

Pro vykreslování postavy dle perspektivy jsem se rozhodl pro kompromis mezi `2D` a `3D`. Tedy postava nebude tvořena 3D modelem, nýbrž obyčejnou 2D texturou, nicméně její umístění bude prováděno pomocí perspektivní projekce. V prostoru se tedy bude umísťovat obdélník s texturou postavy. Perspektiva bude tedy řešena až na úrovni grafické knihovny. Odkud bude brána `3D` pozice bodu, na kterém bude umístěna hlavní postava, se dostanu v následujících odstavcích.

Pro hledání cesty jsem aplikoval řešení pomocí soustavy konvexních polygonů (viz kapitola 1.2.2), které jsem vylepšil následovně:

- sousedící polygony mohou mít více **průchozích bodů** (*tím se zamezí některým nepřirozeně jevícím se cestám*),
- každý polygon bude mít k sobě určenou **3D rovinu**, která bude zajišťovat správnou reprezentaci náklonu a vzdálenost polygonu od pozorovatele – zajištění perspektivy postavy,
- jednotlivé `průchozí body` budou určeny 3D souřadnicemi.

3. Realizace

Program jsem tvořil na OS Linux. Pro samotnou realizaci jsem použil následující aplikace a nástroje:

Vývojové prostředí `NetBeans IDE` – opensourcové prostředí pro vývoj, ladění a distribuci aplikací, podporující mnoho programovacích jazyků. Zvlášť zaměřené je na programovací jazyk **Java**.

Programovací jazyk `Java` – objektově orientovaný programovací jazyk, vyvinutý firmou **Sun Microsystems**. V současnosti je to jeden z nejpoužívanějších programovacích jazyků, je přenositelný mezi různé platformy.

Verzovací systém `GIT` – distribuovaný systém správy verzí, původně vytvořený Linusem Torvaldsem[1].

Herní knihovnu `LWJGL` – řešení zaměřené pro profesionální a amatérské Java programátory k sepisování her komerčních kvalit. Umožňuje přístup k vysokovýkonným meziplatformním knihovnám jako **OpenGL** nebo **OpenAL**.

Grafickou knihovnu `OpenGL` – průmyslový standard specifikující multiplatformní rozhraní pro tvorbu aplikací počítačové grafiky.

Pomocnou knihovnu `Slick-Util` – nahrávání souborů různých grafických formátů a fontů pro použití s **OpenGL**.

`Slick-Util` je jen část knihovny **Slick2D**, což je jednoduchá knihovna pro tvorbu 2D her postavená na **LWJGL**. Funkce z balíku Slick-Util však mají širší využití než jen pro 2D hry, proto byla tato část z knihovny vydána samostatně.

Nyní rozeberu jednotlivé části realizace problematiky, mezi něž patří zejména **herní logika** a **grafická reprezentace**.

[1] Linus Torvalds (* 28. prosince 1969) – finský programátor, známý především zahájením vývoje jádra OS Linux.

3.1 Herní logika

Herní logika je tvořena `objekty` (*předměty, postavy, východy*), `logickými uzly` a `interakcemi`.

3.1.1 Třída logický uzel – LogicNode

Třída **LogicNode** je klíčový prvek herní logiky. Právě pomocí této třídy lze konstruovat logickou strukturu, kde každý uzel může mít:

- Libovolný počet uzlů, na kterých závisí.
- Libovolný počet závisejících uzlů.

Díky této vlastnosti se lze přizpůsobit jakékoli logické situaci. Jakmile je logický uzel splněn, informuje o tom nadřazené uzly, které si ověří, zda-li jsou splněny všechny jejich závislosti. Pakliže všechny závislosti jsou splněny, je i tento uzel splněn a opět o tom informuje nadřazené uzly, pokud nějaké má. Takhle se rekurzivně splní všechny uzly, které mají splněné závislosti.

3.1.2 Třída objekt – Object

Třída **Object** je abstraktním[2] předkem všech typů objektů, mezi které patří:

- `Item` – předmět.
- `CollectableItem` – předmět, který lze sebrat.
- `Person` – postava.
- `Exit` – východ.

Výčet vlastností tohoto předka je následující:

Name určuje jméno objektu.

Title je spjatý s **LogicNode**, což znamená, že obsahuje pole různých titulků vázaných na logický uzel, přičemž při dotazu na titulek objektu se vybere první takový, který má splněný logický uzel. Výchozím titulkem je jméno objektu, takže pokud se žádný titulek nenastaví, použije se jméno.

[2] Abstraktní třída – slouží pouze jako předek objektů, který definuje některé společné metody. Třídu samotnou však není možné instanciovat [Sch12, s. 292].

Titulek objektu znamená řetězec, který se ukáže při najetí myší na daný objekt. Důvod spojení s LogicNode je kvůli tomu, aby bylo možné pojmenovávat předměty podle stavu, ve kterém se nacházejí (*např. „špinavý ubrousek“ a „čistý ubrousek“*).

Place je místo (*tedy lokace*), ve které je objekt aktuálně umístěn. **Inventář** je brán jako speciální případ lokace, která může obsahovat pouze předměty.

Activator a Deactivator jsou spjati s **LogicNode** a slouží pro aktivaci či deaktivaci objektu v závislosti na splnění příslušného logického uzlu.

Interaction respektive InteractionWith jsou vlastnosti, které určují, jaké interakce se mají provést po kliknutí na objekt respektive po použití objektu s odpovídajícím předmětem.

Ony potomci této třídy potom slouží pro rozlišení jednotlivých objektů v lokaci, čímž je hráči umožněno rozlišit mezi sebou tyto objekty. Tak lze dát najevo například rozdílnými kurzory myši. Druhým již zmíněným důvodem pak je odlišení **CollectableItem** od obyčejných předmětů, čímž se rozlišuje, zda-li lze objekt sebrat.

3.1.3 Třída interakce – Interaction

Třída pro interakci byla koncipována tak, aby umožňovala větvení podle aktuálních podmínek. Podmínky jsou svázány s **LogicNode** (logickým uzlem, viz kapitola 3.1.1), podle čehož se vybírá aktuální správná větev – vybere se první větev, která má **splněný** logický uzel.

Jako demonstrace může posloužit např. interakce otevření dveří.

1. Pokud jsou dveře zamčené, proběhne pouze konverzace s cílem informovat hráče, že dveře otevřít nelze.
2. Jsou dveře odemčené, dojde skutečně k jejich otevření (*splní se jiný logický uzel, změní se textura…*).

Jednotlivé větve je možné libovolně vnořovat, lze tedy např. podle jedné podmínky nejprve rozdělit na 2 hlavní větve, které posléze podle další podmínek dělit na další části. Toho lze rovněž využít, pokud mají některé větve část interakce společnou (*např. – využiji-li předchozí příklad s dveřmi – při pokusu o otevření dveří se uvolní a odpadne klika, ať už dveře jsou, či nejsou odemčené*).

Při vytváření interakce je větvení zajištěno metodami `addCondition`, `elseCondition` a `endCondition`. Metoda `addCondition` zahájí větvení – následující akce se budou provádět jen pod určitou podmínkou, která je určena argumentem této metody. Každé další použití metody `addCondition` zahájí větvení do hloubky (*větví aktuální větev*).

Pro zahájení paralelní větve se tedy používá metoda `elseCondition`. Ta vytvoří větev, která se provede v případě, že jsou nesplněné podmínky u předchozích větví, pokud ovšem je buď splněna podmínka této větve nebo tato větev ani žádnou podmínku nemá (*podmínka u metody* ***`elseCondition`*** *je nepovinná*).

Metoda `endCondition` ukončí aktuální větvení a posune se do nadřazené větve. Pokud žádná nadřazená větev neexistuje, vyhodí chybu.

V jednotlivých větvích pak lze používat následují metody:

- `setMessage` – slouží pro rychlé nastavení krátkého rozhovoru, jehož se účastní jen hlavní postava (*komentáře, co hlavní postava provádí*),
- `setTitle` – nastavení titulku akce při najetí kurzorem myši nad objekt, který onu interakci vyvolává,
- `toDelete` – přidání předmětů, které se mají smazat,
- `toPlace` – předměty, které se mají umístit do lokace,
- `toInventoryAdd` – předměty, které se mají přidat do inventáře,
- `setToGo` – nastavení lokace, do které se má přemístit hlavní postava,
- `setGameFinish` – ukončení hry (*hráč dohrál hru*),
- `toProceed` – splnit určené **logické uzly**,
- `toSetOff` – nastavit určené **logické uzly** jako nesplněné,

Ukázka sestavení interakce

```
Interaction interaction = new Interaction()
  .setTitle("Otevrit dvere")
  .addCondition("Polozene prkno")
    .addCondition("Otevreny vrak")
      .setTitle("Podivat se dovnitr")
      .setMessage("A jeje, tohle by se treba mohlo hodit,"
```

```
        + "stare, ale zachovale padlo.",
          "Vypada to, ze nic jineho zde neni.")
        .toInventoryAdd("Padlo")
        .toProceed("Padlo")
      .elseCondition("Odemceny vrak")
        .toProceed("Otevreny vrak")
      .elseCondition()
        .setMessage("A sakris. Dvere do lode jsou zamcene. Ackoliv"
        + "zbytek lodi vypada opravdu ztrouchnivele, zda se,"
        + "ze do kabiny se jen tak nedostanu.")
        .toProceed("Zamceny vrak")
      .endCondition()
    .elseCondition()
      .setMessage("Hm, tak tam asi nedojdu zkrz tu diru v podlaze.")
;
```

Na ukázce je naznačeno, že třída podporuje „plynulé rozhraní“ **Fluent Interface**[3], jednotlivé metody lze tedy volat za sebou a spojovat pouze „tečkou“. Větvení je zde znázorněno odsazováním, pro lepší čitelnost, co se ve které větvi děje, přestože grafická reprezentace nemá žádný vliv na funkčnost kódu.

Dále stojí za povšimnutí, že podmínky – **logické uzly** – jsou zde reprezentovány pouhými řetězci. Je tomu tak z důvodu přehlednějšího zápisu, metody jsou přetíženy tak, aby neakceptovali pouze objekt typu **LogicNode**, ale také řetězce. Metodám jsou pak `logické uzly` zpřístupněny z globálního úložiště, ze kterého se dají jednoznačně identifikovat právě pomocí zástupného řetězce.

3.1.4 Třída konverzace – Conversation

Tato třída se stará o vytvoření a uskutečňování rozhovorů. Umožňuje přidávání jednotlivých **útržků řeči** – `třída Talk` – a **dialogů** – `třída Dialog`. Ke každému útržku řeči je přidána postava, která onen útržek pronáší.

Při vytváření konverzace se používají metody `addTalk`, `addDialog`, `keepUntil` a `toProceedInteraction`. Dokud není použita metoda `addDialog`, zadávané útržky řeči se přidávají do výchozího dialogu, který proběhne při každém uskutečnění dané konverzace.

Parametry metody `addTalk` jsou:

- **postava**, která pronáší útržek – pokud není postava zadána, útržek přísluší

[3] Fluent Interface – „řetězcovité volání“, které umožňuje spojování metod jednoho či více objektů a usnadňuje tak předávání pracovního kontextu objektu.

hlavní postavě,

- **textový řetězec** představující konkrétní útržek řeči.

V této práci jsem neřešil žádné audio stopy, tudíž všechny rozhovory jsou reprezentovány pouze textem.

Parametry metody `addDialog` jsou:

- **název dialogu** – prostý textový řetězec, reprezentující název,
- **logický uzel** – nepovinný parametr **LogicNode**, je-li zadán, dialog se objeví až po splnění logického uzlu.

Metoda `toProceedInteraction` slouží pro provedení určité interakce, která se váže na daný dialog. Parametrem je tedy třída **Interaction**. Kdežto metoda `keepUntil`, jejímž parametrem je opět třída **LogicNode**, ponechává jako aktivní daný dialog, dokud není logický uzel splněn.

Ukázka vytváření konverzace

```
Conversation conversation = new Conversation()
  .addTalk("Dobry den")
  .addTalk("Indian", "Dobry den, pane.")
  .addDialog("Seznameni")
    .addTalk("Ja se jmenuji Karel, jak se jmenujete vy?")
    .addTalk("Indian", "Ja se jmenuji Kucicvikukucu, pane.")
    .addTalk("Co je to proboha za jmeno?")
    .addTalk("Indian", "Co by to bylo za jmeno, pane."
    + "To je jedno z nejbeznejsich jmen.")
    .addTalk("Aha.")
  .addDialog("Pobyt")
    .addTalk("A jak dlouho uz zde pobyvate?")
    .addTalk("Indian", "Moc moc dlouho, pane, moc moc dlouho.")
    .addTalk("Aha.")
  .addDialog("Padlo")
    .toProceedInteraction(new Interaction().toProceed("Vrak"))
    .addTalk("ěChtl bych se zeptat, řpotebovat bych nejake padlo,"
    + "nevite o necem?")
    .addTalk("Indian", "Nevim, nevim, ale zkusil bych nejak prohledat"
    + "tamhleten vrak, nikdy nevite, co se tam muze shovavat...")
```

```
.addDialog("Klic", "Zamceny vrak")
  .keepUntil("Ma musli")
  .toProceedInteraction(new Interaction().toProceed("Dam musli"))
  .addTalk("Zkousel jsem se divat do toho vraku, ale je zamceny,"
  + "nevite nahodou o nejakem klici.")
  .addTalk("Indian", "Klic, nevim, pane, nevim, ale jsem moc smutny,"
  + "protoze jsem ztratil svuj nahrdelnik.")
.addDialog("Klic", "Ma musli")
  .addTalk("Zkousel jsem se divat do toho vraku, ale je zamceny,"
  + "nevite nahodou o nejakem klici.")
  .addTalk("Indian", "No vidite, pane, zrovinka si vzpominam,"
  + "ze ten klic mam vlastne v kapse. Tady mate.")
  .toProceedInteraction(new Interaction().toInventoryAdd("Klic"))
;
```

Na ukázce je opět vidět, že je použito „plynulé rozhraní“, útržky řeči u jednotlivých dialogů jsou pro přehlednost odsazeny. U metod `addTalk`, `addDialog` a `keepUntil` je opět patrné, že jsou použity jejich přetížené varianty, jelikož jsou používány pomocí textového řetězce namísto instance třídy **LogicNode**.

3.2 Grafická implementace

Pro vývoj grafické části práce jsem zvolil **LWJGL** knihovnu pro herní vývojáře, jelikož:

- je to v podstatě jediná a rozhodně nejpodporovanější komplexní sada knihoven pro vývoj her – obsahuje knihovny pro vykreslování (*OpenGL*), přehrávání zvuku (*OpenAL*), řízení vstupů (*klávesnice, myš, joystick…*) a jiné již méně významné části,
- je dostupná pod licencí BSD[4].

Jak již bylo zmíněno, pro vykreslování jsem použil grafickou knihovnu **OpenGL**. Práce s touto knihovnou byla pro mě v této práci nejvíce vzdělávající, jelikož jsem se s ní nikdy předtím nesetkal. Po nastudování některých základních principů [TTSRH02] jsem dospěl dospěl k finálnímu způsobu vykreslování:

[4]BSD licence – licence pro svobodný software, umožňuje šíření licencovaného obsahu, vyžaduje pouze uvedení autora a informaci o licenci.

1. pro vykreslování pozadí lokace a jiných objektů používám ortografickou projekci[5],

2. hlavní postavu vykresluji pomocí perspektivní projekce[6].

Knihovnu `LWJGL` jsem doplnil o knihovnu **Slick-Util**, protože obsahuje nástroje pro načítání textur a fontů z běžných souborových formátů, které nebyly součástí `LWJGL` – `Slick-Util` byla přímo doporučována vývojáři `LWJGL` [Lwjgl02].

3.2.1 Ovládací prvky

Aby bylo možné zpracovávat komunikace hráče s hrou, podle souřadnic kurzoru myši se určuje, není-li na některý objekt ukazováno. Každý objekt disponuje funkci `resolveCollision`, která v případě kolize myší s objektem vrací číslo, které vyjadřuje **z-index** (*hloubku objektu v prostoru*). Pokud lze na jedné pozici nalézt více objektů (*typicky např. šroubovák, který leží na stole*), podle *z-index* se vybere žádaný objekt.

Dalším ovládacím prvkem je pohyb hlavní postavy po lokaci. Místa, kam se může postava pohybovat, jsou označeny konvexními polygony, kde každý polygon má přidruženou *3D rovinu* (viz kapitola 2.2). Podle pozice kurzoru myši lze zvolit konkrétní polygon, avšak nelze tak získat 3D pozici postavy v prostoru. Pro tento účel má *OpenGL* implementovanou funkci čtení **3D souřadnic** bodu, na který ukazuje bod ležící na **x, y** souřadnicích pohledu.

Vykreslí se tedy průhlednou barvou v perspektivní projekci polygon představující onu *3D rovinu*, která nese informaci o tvaru zobrazovaného prostoru, a podle koordinací pozice myši se zjistí **3D bod** v prostoru, do kterého má hlavní postava dojít.

3.2.2 Vykreslování

Vykreslování probíhá v několika krocích:

1. Nejprve se vykreslí hlavní pozadí lokace.

[5] Ortografická projekce – pravoúhlá projekce 3D těles do dvou rozměrů, kde jsou všechny projekční paprsky navzájem kolmé.

[6] Perspektivní projekce – všechny projekční paprsky jsou svedené do jednoho bodu, vzdálenější objekty se tak zdají být menší než objekty bližší.

2. Vymaže se hloubkový buffer[7] a vykreslí se další vrstvy pozadí lokace – ty, které jsou zobrazeny aktuálně za postavou.

3. Vykreslí se všechny objekty (*předměty, postavy…*), které jsou aktuálně za hlavní postavou.

4. Vymaže se hloubkový buffer, přepne se do perspektivní projekce a vykreslí se hlavní postava.

5. Opět se vymaže hloubkový buffer, přepne se zpátky do ortogonální projekce a vykreslí se zbytek objektů.

6. Nakonec jsou vykresleny i zbylé vrstvy lokace.

Knihovna *OpenGL* si při vykreslování ukládá údaje o jejich hloubce, což poté využívá při řešení viditelnosti jednotlivých objektů. Pokud je tedy potřeba překreslit nějaké objekty přes již vykreslené nehledě na jejich hloubce v prostoru, je potřeba vymazat *hloubkový buffer*.

3.2.3 Vykreslení textu

Pro vykreslování fontů není v knihovně *OpenGL* zabudováno žádné implicitní řešení, existují však různé knihovny, které toto řeší. Já jsem zvolil nástroje knihovny `Slick-Util`, které umí načítat a používat přímo fonty v souborech `TTF`[8], neboť jsem tuto knihovnu používal již pro načítání textur.

Nejedná se však o ideální řešení, jelikož je vykreslování touto metodou poměrně pomalé. Rychlost vykreslování šla ještě níž po té, co jsem řešil problém s viditelností textu na podkladu jakékoli barvy. Tento problém jsem se rozhodl řešit vykreslením rámečku kolem znaků kontrastní barvou, pro vykreslení rámečku však neexistují žádné metody nabízené použitou knihovnou. Rámeček tedy tvořím tak, že celý text vykreslím 4krát s posunutím do všech směrů jako podklad a na skutečnou pozici potom vykreslím onen text. Toto řešení má však za následek, zvláště u vykreslování větších bloků textu, výrazné snímkové frekvence, které z naměřených hodnot činilo snížení až o 60 %.

[7] Hloubkový buffer – paměť, ve které jsou ukládány údaje o hloubce jednotlivých vygenerovaných pixelů.

[8] TrueType – standard pro popis vektorových fontů

3.2.4 Konzolová reprezentace

Jako odbočku bych ještě krátce představil reprezentaci pomocí **konzolové aplikace**[9]. Poslouží jako ukázka oddělené logiky od grafické reprezentace, lze tedy vytvořit jednoduchou hru i v textovém režimu.

Pro konzolovu aplikaci se neřeší žádné překrývání objektů, kolize či pohyb postavy, interakce s hráčem probíhá formou příkazů. Hráč má k dispozici příkazy pro výpis *objektů z aktuální lokace* a výpis *předmětů z inventáře*. Dalšími příkazy pak zkoumá či používá jednotlivé objekty nebo používá předměty z inventáře na ony objekty.

Ukázka konzolové reprezentace

```
Game start
:l
-- Les --
-x- K mori
 +  Pazourek
 #  Strom
 +  Stan
 #  Zip
 +  Provazek
 +  Klacek
-x- Do stanu
==================
:Pazourek
Ostry a tupy kamen, tim by se dalo leccos prestipnout.
:Klacek
A hele, docela pevny kus vetve, to by se mohlo hodit.
:i
-- Inventar --
 +  Pazourek
 +  Klacek
==================
:Pazourek > Klacek
A hele, to je ale pekna sekyrka z pouheho klacku a kusu kamene!
:q
```

[9]Konzolová aplikace – počítačový program, jenž nevyužívá grafické rozhraní, ale systémovou konzoli či terminál.

V ukázkovém příkladu konzolové aplikace jsou použity následující příkazy:

- „**l**“ – výpis objektů z lokace,
- „**i**“ – výpis předmětů z inventáře,
- „[**jméno objektu**]“ – zkoumání objektu,
- „[**jméno předmětu z inventáře**] > [**jméno objektu**]“ – použití předmětu na objekt,
- „**q**“ – ukončení aplikace.

4. Příklad

Posledním bodem zadání této práce bylo vytvoření příkladu, který by demonstroval použití vytvořených algoritmů. Pro tento úkol jsem vytvořil krátkou hru s jednoduchým dějem, který v podstatě ani nemá příběh – slouží jen pro ukázku použití všech vlastností a rysů.

„Děj“ se tedy odehrává kdesi na pustém ostrově, není však příliš důležitý. Vytvořená aplikace nemá ani menu – po spuštění se hra spustí a po dokončení hry se program zase ukončí. Ovládání hry je následující:

Levým tlačítkem myši se uskutečňují veškeré akce – ovládá se jím hlavní postava. Pokud se klikne na:

- nějaký **objekt**, uskuteční se s ním spojená interakce,
- nějaké **místo**, kde se může postava pohybovat, postava se vydá vstříc onomu místu.

Pravým tlačítkem myši se zobrazuje respektive skrývá inventář –ten je možné vyvolat rovněž *mezerníkem*.

Herní obrazovka je uzpůsobena poměru stran **16:9**, pokud tedy monitor nedisponuje tímto poměrem, příslušná část se vyplní černými pruhy. Inventář se zobrazuje hned nahoře jako jeden proužek. Popisky k objektům a interakcím se vykreslují u kurzoru myši, jednotlivé dialogy konverzace se vykreslují v levé dolní části herní obrazovky. Pro lepší představu je v příloze (na straně 44) snímek přímo ze spuštěné hry.

Závěr

Úkolem této práce bylo vytvořit sadu algoritmů, které se používají v `point and click adventure` hrách. To bylo splněno minimálně do takové míry, že pomocí nich lze zkonstruovat funkční jednoduchou hru – pro sestavování složitějších her by bylo zapotřebí nejprve vytvořit komplexní sadu programů, jenž by vystupovaly jako editory, ve kterých by bylo možné interaktivně vytvářet jednotlivé prvky hry (*aplikace textur na jednotlivé objekty a lokace, vyznačování pozic objektů, vyznačování oblastí pro pohyb postavy…*).

Mezi věci, které by stálo za to vylepšit, patří určitě vykreslování fontů, jelikož je opravdu pomalé. V jednoduchých hrách bez použití náročnějších animací by to nebylo tak znatelné, ale pro větší projekty by byl současný způsob vykreslování fontů nedostačující. Co se týče věcí, které jsem v této práci neřešil, patří zde přehrávání audio stop, které jsou pro hry tohoto žánru, ostatně jako pro jakékoli jiné hry, klíčovou – ne však nezbytnou – součástí. Jedná se o přehrávání zvukových efektů, hudbu nebo pro tyto hry typický dabing dialogů.

Rozšiřování této práce by se tedy týkalo zejména chybějícího audia a oprava způsobu vykreslování fontů. Mezi další rozšiřování patří zakomponování některých dalších funkcionalit, které by však vyplynuly z praxe, nebo úprava a optimalizace pro umožňění spouštění na mobilních zařízeních se systémem Android.

Literatura

[Fin07] Alien Ryder Flex. FINLEY, Darel Rex. *Determining Whether A Point Is Inside A Complex Polygon* [online]. 2007 [cit. 2013-05-02]. Dostupné z: http://alienryderflex.com/polygon/

[Lwjgl02] *Lightweight Java Game Library Project* [online]. 2002 [cit. 2013-04-11]. Dostupné z: http://www.lwjgl.org/

[Sch12] SCHILDT, Herbert. *Java 7: výukový kurz*. 1. vyd. Brno: Computer Press, 2012, 664 s. ISBN 978-80-251-3748-2.

[Tich11] TICHÁČEK, Petr. Historie adventur #1: jak to všechno začalo. *GAMES.CZ* [online]. 2011 [cit. 2013-05-04]. Dostupné z: http://games.tiscali.cz/tema/historie-adventur-1-jak-to-vsechno-zacalo-56294

[TTSRH02] TUREK, Michal, Milan TUREK, Václav SLOVÁČEK, Jiří RAJSKÝ, Pavel HRADSKÝ, Bernard LIDICKÝ, Marek OLŠÁK, Přemysl JAROŠ a Radomír VRÁNA. *CZ NeHe OpenGL* [online]. 2002 [cit. 2013-04-17]. Dostupné z: http://nehe.ceske-hry.cz/tut_obsah.php

Seznam příloh

A – Snímek zachycující běh příkladové aplikace.

B – CD, na kterém se nachází:

- elektronická podoba zprávy,
- zdrojové kódy.

A Snímek zachycující běh příkladové aplikace